

# Trabajo 3

Antonio Álvarez Caballero

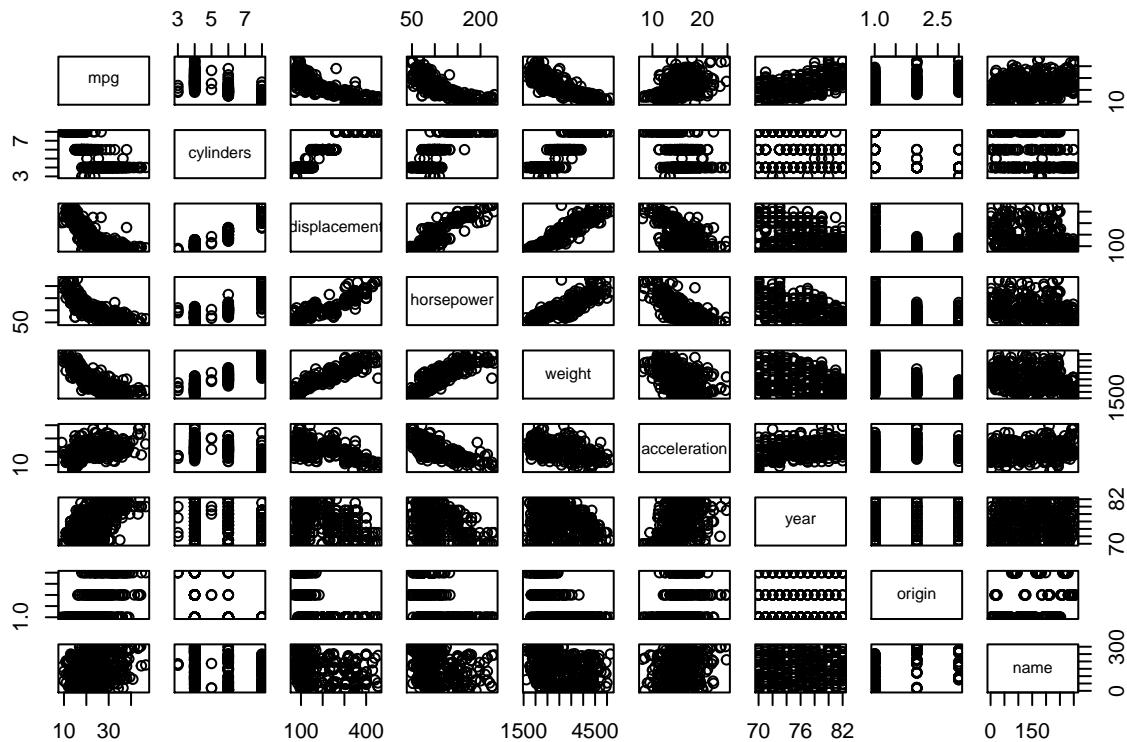
27 de mayo de 2016

## Ejercicio 1

```
## The following object is masked from package:ggplot2:
```

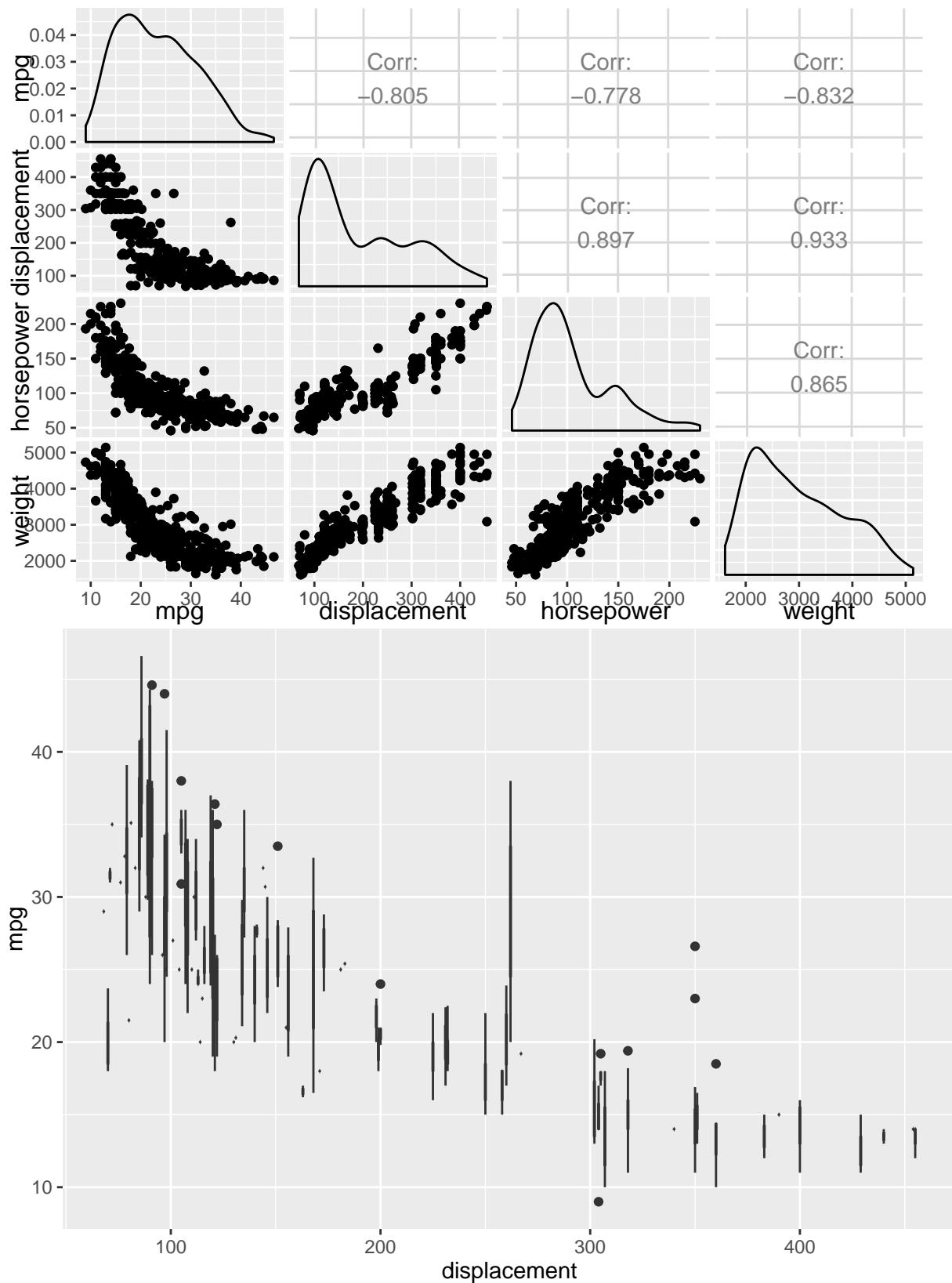
```
##
```

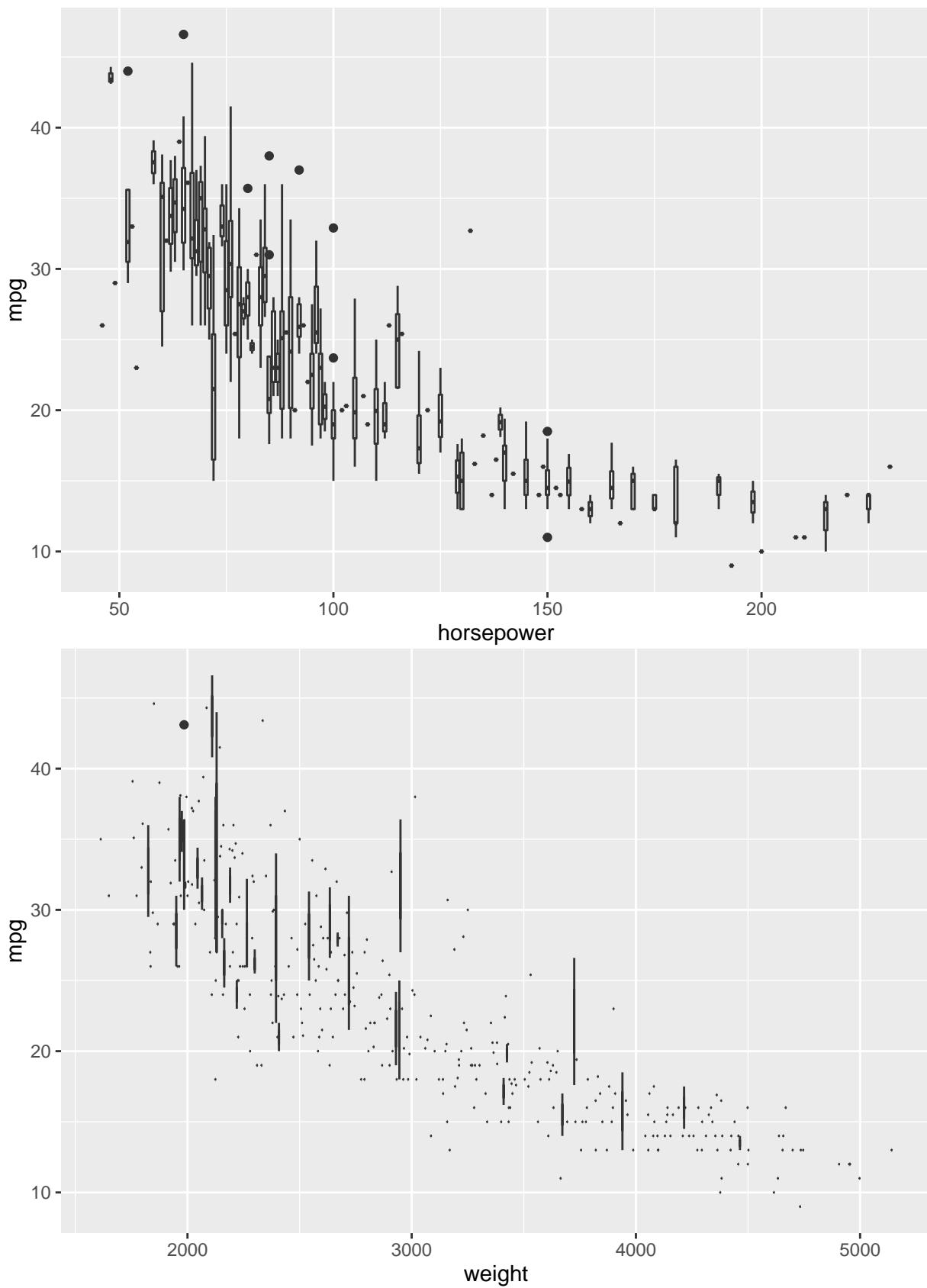
```
##     mpg
```



### Apartado a)

Parece ser que las variables de las que más depende *mpg* son *displacement*, *horsepower* y *weight*. Veámoslas con más detalle.





Es claro que existe una dependencia entre estas variables. Además, con el primer plot podemos ver las

correlaciones entre estas 3 variables, que es alta.

## Apartado b)

Seleccionamos las variables que hemos decidido para predecir.

```
Auto.selected <- Auto[,c("displacement","horsepower","weight")]
```

## Apartado c)

Como nuestro conjunto de datos es grande (392 instancias), podemos realizar un muestreo aleatorio. Así tampoco falseamos las muestras, cosa que podría pasarnos si realizamos un muestreo estratificado.

```
index <- sample(nrow(Auto), size = 0.8*nrow(Auto) )  
  
Auto.train <- Auto.selected[index,]  
Auto.test <- Auto.selected[-index,]
```

## Apartado d)

Vamos a crear una nueva variable, *mpg01*, la cual tendrá 1 si el valor de *mpg* está por encima de la mediana y -1 en otro caso.

```
mpg01 <- ifelse(Auto$mpg >= median(Auto$mpg), 1, 0)  
Auto.selected$mpg01 <- mpg01  
Auto.train$mpg01 <- mpg01[index]  
Auto.test$mpg01 <- mpg01[-index]
```

### Apartado d1)

Vamos a ajustar un modelo de regresión logística para predecir *mpg01*.

```
model.LogReg <- glm(mpg01 ~ ., data = Auto.train, family = binomial)  
prediction.LogReg <- predict(model.LogReg, newdata = Auto.test, type = "response")  
prediction.LogReg.labels <- ifelse(prediction.LogReg > 0.5, 1, 0)  
error.test <- "¿Esto cómo es?"  
error.test <- sum(sign(prediction.LogReg.labels) != sign(Auto.test$mpg01)) /  
length(prediction.LogReg.labels)
```

El error de test de este modelo es 8.8607595.

### Apartado d2)

Ahora vamos a ajustar un modelo k-NN.

### Apartado d3)

Veamos las curvas ROC de ambos modelos.

```

roc.prediction.regLog <- prediction(prediction.LogReg, Auto.test$mpg01)
roc.performance.regLog <- performance(roc.prediction.regLog, measure = "tpr", x.measure = "fpr")

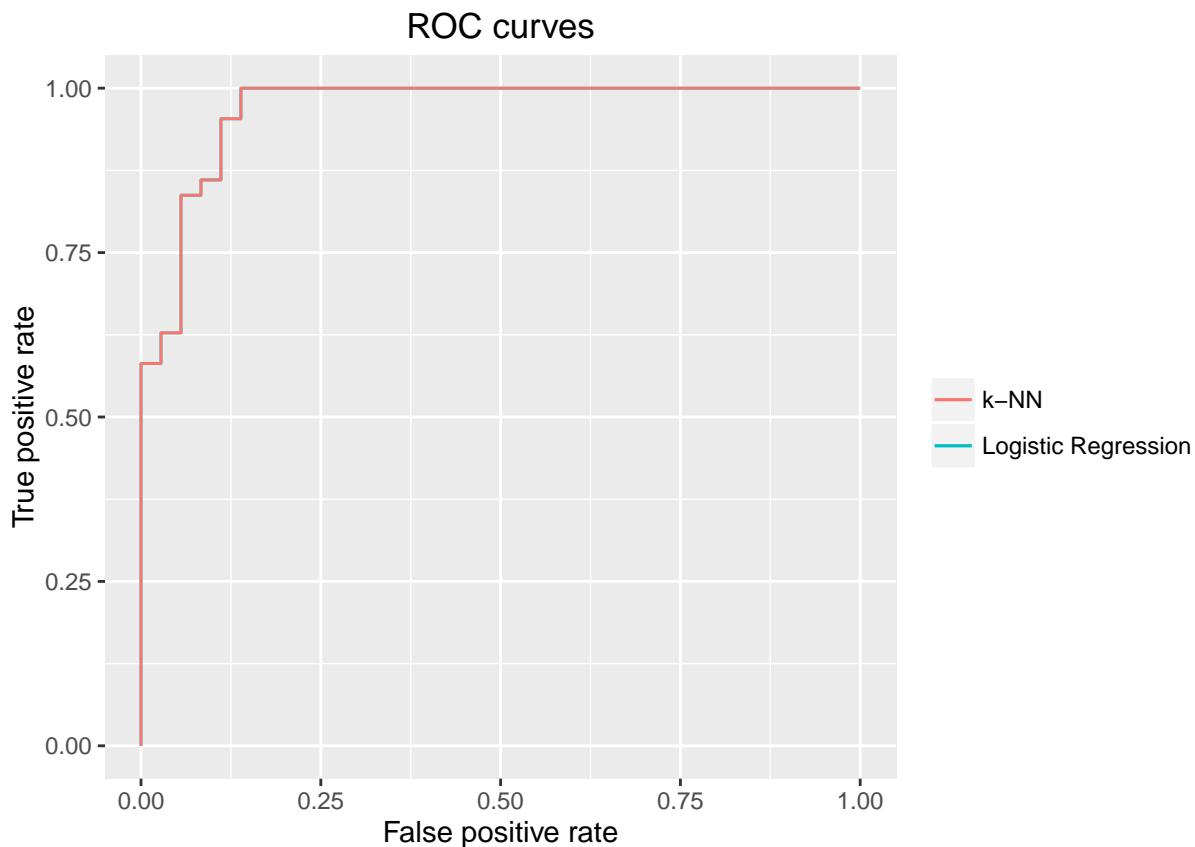
### Meter del knn

#####
roc.performance.knn <- roc.performance.regLog

roc.data <- data.frame(x = roc.performance.regLog@x.values[[1]],
                       y1 = roc.performance.regLog@y.values[[1]],
                       y2 = roc.performance.knn@y.values[[1]])

ggplot(roc.data, aes(x)) +
  geom_line(aes(y = y1, colour = "Logistic Regression")) +
  geom_line(aes(y = y2, colour = "k-NN")) +
  theme(legend.title = element_blank()) +
  labs(title= "ROC curves", x = "False positive rate", y = "True positive rate")

```



```

auc.regLog <- auc(roc.data$x,roc.data$y1, type = 'spline')
auc.knn <- auc(roc.data$x,roc.data$y2, type = 'spline')

```

El área bajo la curva de la ROC de regresión logística es 0.9651519 y la del k-NN es 0.9651519. Luego k-NN es el modelo que mejor *performance* tiene.

## Apartado e) (Bonus-1)

Para estudiar el error con validación cruzada hacemos uso de `cv.glm`

```
model.full.LogReg <- glm(mpg01 ~ ., data = Auto.selected)
cv.LogReg <- cv.glm(data = Auto.selected, glmfit = model.full.LogReg, K = 5)
cv.LogReg$delta
```

```
## [1] 0.1043928 0.1040979
```

El error estimado es el primero de este vector. El segundo es un ajuste para compensar el sesgo introducido al no usar *Leave-One-Out*.

Para el caso del k-NN

Por tanto, vemos que es mejor *uno*.

## Apartado f) (Bonus-2)

Por hacer

## Ejercicio 2

### Apartado a)

Ajustamos con validación cruzada sobre la variable *crim*, que es la que está en la posición 1.

```
attach(Boston)
set.seed(123456789)

index <- sample(nrow(Boston), 0.8*nrow(Boston))
Boston.full <- Boston
Boston.train <- Boston[index,]
Boston.test <- Boston[-index,]

model.Boston <- glmnet(as.matrix(Boston.train[,-1]), Boston.train[,1], alpha = 1)
```

### Apartado b)

Ahora utilizamos un método LASSO y seleccionamos las variables que están por encima de un umbral.

```
cv.Boston <- cv.glmnet(as.matrix(Boston[,-1]), Boston[,1], nfolds = 5, alpha = 1)
lasso.coeff <- predict(cv.Boston, type="coefficients", s = cv.Boston$lambda.min)
threshold <- 0.1
selected <- which(abs(lasso.coeff) > threshold)[-1]
```

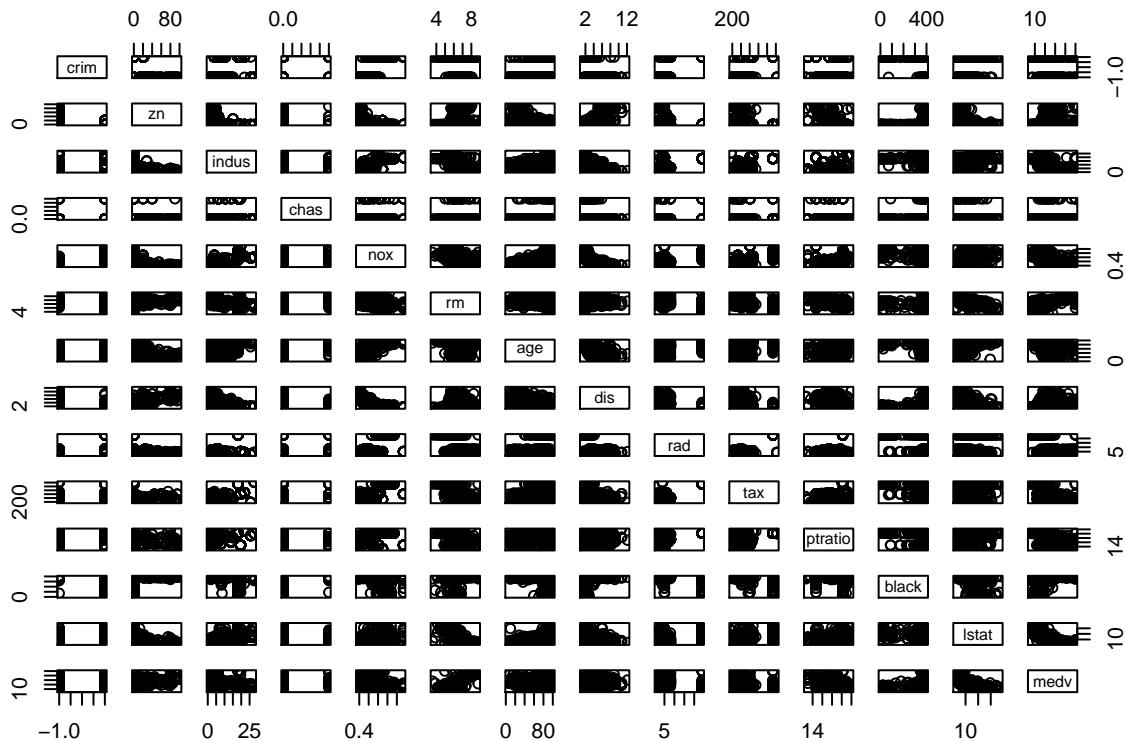
Con esto afirmamos que las características que superan nuestro umbral 0.1 son 4, 5, 6, 8, 9, 11, 13, 14.

Seguir con regularización.

## Apartado c)

Al igual que en el anterior apartado, definimos una nueva variable usando la mediana como umbral.

```
Boston.full$crim <- ifelse(Boston$crim > median(Boston$crim), 1, -1)
Boston.train.crim1 <- Boston.full[index,]
Boston.test.crim1 <- Boston.full[-index,]
pairs(Boston.full)
```



Ahora ajustamos varias *SVM*, probaremos la lineal y con los núcleos disponibles, y veremos cómo se comporta cada uno. Posiblemente el núcleo lineal no sea suficiente, ya que los datos no parecen lo suficientemente separados. A priori parece que uno polinomial hará un mejor trabajo.

```
svm.linear <- svm(crim ~ ., data = Boston.train.crim1, kernel = "linear")
svm.linear.prediction <- predict(svm.linear, newdata = Boston.test.crim1[,-1])
confusionMatrix(sign(svm.linear.prediction), Boston.test.crim1$crim)
```

```
## Confusion Matrix and Statistics
##
##             Reference
## Prediction -1  1
##           -1 51 10
##            1   3 38
##
##                  Accuracy : 0.8725
##                  95% CI : (0.7919, 0.9304)
##      No Information Rate : 0.5294
##      P-Value [Acc > NIR] : 1.616e-13
##
##                  Kappa : 0.7421
##  Mcnemar's Test P-Value : 0.09609
##
```

```

##           Sensitivity : 0.9444
##           Specificity : 0.7917
##           Pos Pred Value : 0.8361
##           Neg Pred Value : 0.9268
##           Prevalence : 0.5294
##           Detection Rate : 0.5000
##           Detection Prevalence : 0.5980
##           Balanced Accuracy : 0.8681
##
##           'Positive' Class : -1
##
svm.polynomial <- svm(crim ~ ., data = Boston.train.crim1, kernel = "polynomial")
svm.polynomial.prediction <- predict(svm.polynomial, newdata = Boston.test.crim1[,-1])
confusionMatrix(sign(svm.polynomial.prediction), Boston.test.crim1$crim)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##           -1 52  8
##           1   2 40
##
##           Accuracy : 0.902
##           95% CI : (0.8271, 0.952)
##           No Information Rate : 0.5294
##           P-Value [Acc > NIR] : 5e-16
##
##           Kappa : 0.8019
##           Mcnemar's Test P-Value : 0.1138
##
##           Sensitivity : 0.9630
##           Specificity : 0.8333
##           Pos Pred Value : 0.8667
##           Neg Pred Value : 0.9524
##           Prevalence : 0.5294
##           Detection Rate : 0.5098
##           Detection Prevalence : 0.5882
##           Balanced Accuracy : 0.8981
##
##           'Positive' Class : -1
##
svm.radial <- svm(crim ~ ., data = Boston.train.crim1, kernel = "radial")
svm.radial.prediction <- predict(svm.radial, newdata = Boston.test.crim1[,-1])
confusionMatrix(sign(svm.radial.prediction), Boston.test.crim1$crim)

## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##           -1 49  6
##           1   5 42
##
##           Accuracy : 0.8922

```

```

##                               95% CI : (0.8152, 0.9449)
##      No Information Rate : 0.5294
##      P-Value [Acc > NIR] : 3.774e-15
##
##                          Kappa : 0.7833
##  Mcnemar's Test P-Value : 1
##
##      Sensitivity : 0.9074
##      Specificity : 0.8750
##      Pos Pred Value : 0.8909
##      Neg Pred Value : 0.8936
##      Prevalence : 0.5294
##      Detection Rate : 0.4804
##      Detection Prevalence : 0.5392
##      Balanced Accuracy : 0.8912
##
##      'Positive' Class : -1
##

svm.sigmoid <- svm(crim ~ ., data = Boston.train.crim1, kernel = "sigmoid")
svm.sigmoid.prediction <- predict(svm.sigmoid, newdata = Boston.test.crim1[,-1])
confusionMatrix(sign(svm.sigmoid.prediction), Boston.test.crim1$crim)

## Confusion Matrix and Statistics
##
##      Reference
## Prediction -1  1
##       -1 22 18
##        1 32 30
##
##      Accuracy : 0.5098
##      95% CI : (0.4089, 0.6101)
##      No Information Rate : 0.5294
##      P-Value [Acc > NIR] : 0.69044
##
##                          Kappa : 0.0319
##  Mcnemar's Test P-Value : 0.06599
##
##      Sensitivity : 0.4074
##      Specificity : 0.6250
##      Pos Pred Value : 0.5500
##      Neg Pred Value : 0.4839
##      Prevalence : 0.5294
##      Detection Rate : 0.2157
##      Detection Prevalence : 0.3922
##      Balanced Accuracy : 0.5162
##
##      'Positive' Class : -1
##

```

Al final, el

## Apartado d) (Bonus-3)

Ajustamos con validación cruzada sobre la variable *crim*.

```
cv.Boston <- cv.glmnet(as.matrix(Boston[,-1]), Boston[,1], nfolds = 5, alpha = 1)
```

El error de validación cruzada es

```
cv.Boston$cvm
```

```
## [1] 73.44629 69.69340 65.57450 62.15294 59.31050 56.94901 54.98697  
## [8] 53.35530 51.92575 50.57490 49.37295 48.37637 47.54303 46.84977  
## [15] 46.27829 45.80756 45.42499 45.11508 44.85812 44.63284 44.44202  
## [22] 44.28251 44.13463 43.99118 43.87275 43.77272 43.68583 43.61130  
## [29] 43.54863 43.49150 43.41694 43.35179 43.29459 43.22690 43.17175  
## [36] 43.12449 43.08471 43.03839 42.99640 42.94169 42.87580 42.82081  
## [43] 42.76855 42.72311 42.68992 42.66404 42.64521 42.63186 42.62700  
## [50] 42.62509 42.62654 42.62972 42.63239 42.63564 42.63938 42.64245  
## [57] 42.64645 42.65055 42.65177 42.65493 42.65777 42.66085 42.66415  
## [64] 42.66737 42.67058 42.67367 42.67688 42.68009 42.68274 42.68554  
## [71] 42.68788 42.69060 42.69247 42.69449 42.69618 42.69745
```

Completar solución.

## Ejercicio 3

### Apartado a)

Ya tenemos cargado y separado el conjunto de datos en 80% training y 20% test.

### Apartado b)

Vamos a ajustar un modelo de Bagging. Para ello usaremos el RandomForest y le diremos que use el total de características disponibles.

```
bagging <- randomForest(medv ~., data = Boston, subset = -index, mtry = ncol(Boston)-1, importance = TRUE)  
bagging.prediction <- predict(bagging, newdata = Boston.test)  
bagging.error <- "¿Esto cómo es?"
```

El error de test del modelo bagging es ¿Esto cómo es?.

### Apartado c)

Ahora vamos a ajustar un RandomForest.

```
randomFor <- randomForest(medv ~., data = Boston, subset = -index, importance = TRUE)  
  
randomFor.error <- "¿Esto cómo es?"
```

El número de árboles usado es 500. El error de test es ¿Esto cómo es?.

La diferencia con bagging es ...

## Apartado d)

Ajustamos un modelo de regresión con Boosting.

```
boosting <- gbm(medv~., data = Boston.train, distribution = "gaussian")
pred <- predict(boosting, Boston.test, n.trees = 100)
boosting.error <- "¿Esto cómo es?"
```

El error de test es ¿Esto cómo es?.

La diferencia con bagging y randomForest es...

## Ejercicio 4

### Apartado a)

Cogemos una muestra aleatoria de 800 elementos y lo usamos como training.

```
index <- sample(nrow(OJ), 800)
OJ.train <- OJ[index,]
OJ.test <- OJ[-index,]
```

Ajustamos un árbol con la variable *Purchase* como objetivo.

```
model.tree <- tree(Purchase ~ ., data = OJ.train)
```

### Apartado b)

Veamos un resumen del árbol.

```
summary(model.tree)

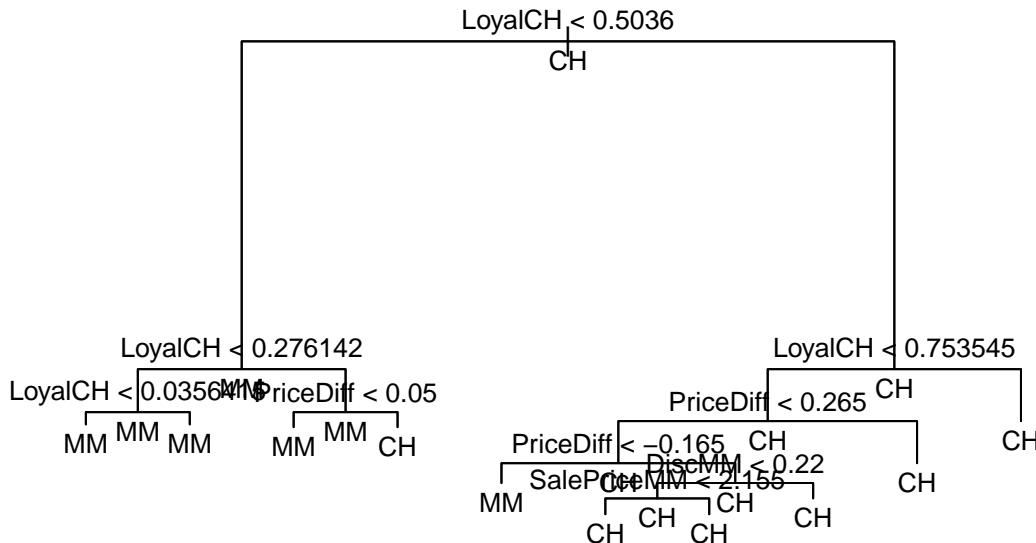
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJ.train)
## Variables actually used in tree construction:
## [1] "LoyalCH"      "PriceDiff"     "DiscMM"        "SalePriceMM"
## Number of terminal nodes:  10
## Residual mean deviance:  0.7312 = 577.7 / 790
## Misclassification error rate: 0.1713 = 137 / 800
```

El número de nodos terminales es de 8, y tiene un error del 16.38%. Completar.

### Apartado c)

Dibujamos el árbol obtenido.

```
plot(model.tree, main="Classification tree")
text(model.tree, all=TRUE, cex=.8)
```



Interpretar.

### Apartado d)

Aplicamos el árbol a nuestros datos de test.

```

prediction.tree <- predict(model.tree, OJ.test)
prediction.tree

```

```

##          CH      MM
## 13  0.95785441 0.04214559
## 16  0.95785441 0.04214559
## 19  0.53846154 0.46153846
## 20  0.53846154 0.46153846
## 27  0.95785441 0.04214559
## 30  0.95785441 0.04214559
## 31  0.53846154 0.46153846
## 43  0.95785441 0.04214559
## 44  0.95785441 0.04214559
## 46  0.95785441 0.04214559
## 48  0.95785441 0.04214559
## 50  0.95785441 0.04214559
## 55  0.95121951 0.04878049
## 67  0.95785441 0.04214559
## 71  0.26666667 0.73333333
## 74  0.95785441 0.04214559
## 78  0.95785441 0.04214559
## 93  0.53846154 0.46153846
## 104 0.95785441 0.04214559
## 105 0.50000000 0.50000000
## 106 0.95785441 0.04214559
## 110 0.95785441 0.04214559
## 111 0.95785441 0.04214559
## 113 0.95785441 0.04214559
## 116 0.95785441 0.04214559
## 122 0.95785441 0.04214559

```

```

## 124 0.95785441 0.04214559
## 133 0.95785441 0.04214559
## 134 0.95785441 0.04214559
## 137 0.95785441 0.04214559
## 141 0.50000000 0.50000000
## 142 0.53846154 0.46153846
## 149 0.18181818 0.81818182
## 151 0.18181818 0.81818182
## 157 0.95785441 0.04214559
## 159 0.95785441 0.04214559
## 160 0.95785441 0.04214559
## 162 0.95785441 0.04214559
## 165 0.95785441 0.04214559
## 169 0.95785441 0.04214559
## 176 0.95785441 0.04214559
## 177 0.95785441 0.04214559
## 178 0.95785441 0.04214559
## 180 0.95785441 0.04214559
## 182 0.95785441 0.04214559
## 183 0.95785441 0.04214559
## 194 0.95785441 0.04214559
## 198 0.95785441 0.04214559
## 199 0.95785441 0.04214559
## 209 0.95785441 0.04214559
## 210 0.95785441 0.04214559
## 212 0.95785441 0.04214559
## 213 0.95785441 0.04214559
## 218 0.95785441 0.04214559
## 220 0.95785441 0.04214559
## 223 0.18181818 0.81818182
## 228 0.53846154 0.46153846
## 231 0.26666667 0.73333333
## 234 0.96000000 0.04000000
## 239 0.95785441 0.04214559
## 240 0.95785441 0.04214559
## 245 0.95785441 0.04214559
## 246 0.95785441 0.04214559
## 247 0.95785441 0.04214559
## 248 0.95785441 0.04214559
## 250 0.95785441 0.04214559
## 256 0.18181818 0.81818182
## 264 0.50000000 0.50000000
## 265 1.00000000 0.00000000
## 267 0.96000000 0.04000000
## 269 0.53846154 0.46153846
## 276 0.01886792 0.98113208
## 279 0.01886792 0.98113208
## 280 0.01886792 0.98113208
## 281 0.01886792 0.98113208
## 291 0.01886792 0.98113208
## 294 0.01886792 0.98113208
## 301 0.53846154 0.46153846
## 302 0.26666667 0.73333333
## 310 0.53846154 0.46153846

```

```

## 311 0.53846154 0.46153846
## 312 0.50000000 0.50000000
## 315 0.95785441 0.04214559
## 318 0.95121951 0.04878049
## 320 0.53846154 0.46153846
## 326 0.53846154 0.46153846
## 330 0.53846154 0.46153846
## 332 0.26666667 0.73333333
## 334 0.18181818 0.81818182
## 335 0.18181818 0.81818182
## 337 0.18181818 0.81818182
## 338 0.18181818 0.81818182
## 342 0.95785441 0.04214559
## 344 0.95785441 0.04214559
## 346 0.95785441 0.04214559
## 349 0.95785441 0.04214559
## 354 0.26666667 0.73333333
## 358 0.53846154 0.46153846
## 359 0.53846154 0.46153846
## 360 0.50000000 0.50000000
## 368 0.18181818 0.81818182
## 369 0.18181818 0.81818182
## 370 0.53846154 0.46153846
## 373 0.53846154 0.46153846
## 376 0.53846154 0.46153846
## 378 0.18181818 0.81818182
## 379 0.18181818 0.81818182
## 385 0.18181818 0.81818182
## 392 0.18181818 0.81818182
## 405 0.18181818 0.81818182
## 410 0.53846154 0.46153846
## 412 0.26666667 0.73333333
## 414 0.18181818 0.81818182
## 425 0.53846154 0.46153846
## 426 0.26666667 0.73333333
## 429 0.53846154 0.46153846
## 431 0.18181818 0.81818182
## 434 0.26666667 0.73333333
## 435 0.26666667 0.73333333
## 436 0.53846154 0.46153846
## 438 0.26666667 0.73333333
## 441 0.95121951 0.04878049
## 442 0.95121951 0.04878049
## 443 0.95121951 0.04878049
## 449 0.50000000 0.50000000
## 452 0.53846154 0.46153846
## 460 0.26666667 0.73333333
## 468 0.95785441 0.04214559
## 475 0.18181818 0.81818182
## 488 0.50000000 0.50000000
## 490 0.95785441 0.04214559
## 493 0.95785441 0.04214559
## 507 0.95785441 0.04214559
## 518 0.95785441 0.04214559

```

```

## 519 0.53846154 0.46153846
## 521 0.96000000 0.04000000
## 525 0.26666667 0.73333333
## 530 0.96000000 0.04000000
## 544 0.18181818 0.81818182
## 549 0.18181818 0.81818182
## 563 0.50000000 0.50000000
## 574 0.18181818 0.81818182
## 577 0.95121951 0.04878049
## 580 0.53846154 0.46153846
## 582 0.95121951 0.04878049
## 603 0.95785441 0.04214559
## 607 0.95785441 0.04214559
## 608 0.95785441 0.04214559
## 609 0.95785441 0.04214559
## 611 0.95121951 0.04878049
## 615 0.96000000 0.04000000
## 618 0.95785441 0.04214559
## 620 0.95785441 0.04214559
## 621 0.95785441 0.04214559
## 632 0.96000000 0.04000000
## 633 0.26666667 0.73333333
## 634 0.95785441 0.04214559
## 637 0.95785441 0.04214559
## 638 0.95121951 0.04878049
## 647 0.95785441 0.04214559
## 649 0.95785441 0.04214559
## 655 0.95785441 0.04214559
## 663 0.53846154 0.46153846
## 666 0.26666667 0.73333333
## 669 0.18181818 0.81818182
## 680 0.26666667 0.73333333
## 682 0.18181818 0.81818182
## 687 0.18181818 0.81818182
## 688 0.18181818 0.81818182
## 696 0.01886792 0.98113208
## 705 0.01886792 0.98113208
## 712 0.01886792 0.98113208
## 717 0.01886792 0.98113208
## 719 0.01886792 0.98113208
## 721 0.01886792 0.98113208
## 722 0.01886792 0.98113208
## 723 0.01886792 0.98113208
## 725 0.01886792 0.98113208
## 726 0.01886792 0.98113208
## 727 0.01886792 0.98113208
## 731 0.26666667 0.73333333
## 732 0.26666667 0.73333333
## 737 0.26666667 0.73333333
## 738 0.53846154 0.46153846
## 739 0.53846154 0.46153846
## 744 0.26666667 0.73333333
## 747 0.26666667 0.73333333
## 748 0.18181818 0.81818182

```

```

## 755 0.26666667 0.73333333
## 756 0.53846154 0.46153846
## 757 0.53846154 0.46153846
## 761 0.53846154 0.46153846
## 762 0.50000000 0.50000000
## 766 0.95785441 0.04214559
## 772 0.53846154 0.46153846
## 775 0.18181818 0.81818182
## 777 0.96000000 0.04000000
## 781 0.95121951 0.04878049
## 782 0.50000000 0.50000000
## 796 0.18181818 0.81818182
## 799 0.50000000 0.50000000
## 806 0.95121951 0.04878049
## 807 0.95121951 0.04878049
## 818 0.95785441 0.04214559
## 819 0.95785441 0.04214559
## 825 0.95785441 0.04214559
## 831 0.95785441 0.04214559
## 838 0.26666667 0.73333333
## 840 0.18181818 0.81818182
## 841 0.18181818 0.81818182
## 842 0.53846154 0.46153846
## 846 0.18181818 0.81818182
## 850 0.53846154 0.46153846
## 856 0.95785441 0.04214559
## 858 0.95785441 0.04214559
## 859 0.95785441 0.04214559
## 867 0.96000000 0.04000000
## 871 0.96000000 0.04000000
## 874 0.53846154 0.46153846
## 878 0.95785441 0.04214559
## 881 0.95785441 0.04214559
## 886 0.26666667 0.73333333
## 888 0.95121951 0.04878049
## 891 0.53846154 0.46153846
## 892 0.95121951 0.04878049
## 902 0.95785441 0.04214559
## 904 0.95785441 0.04214559
## 906 0.18181818 0.81818182
## 910 0.53846154 0.46153846
## 913 0.53846154 0.46153846
## 924 0.26666667 0.73333333
## 925 0.26666667 0.73333333
## 926 0.26666667 0.73333333
## 931 0.18181818 0.81818182
## 934 0.26666667 0.73333333
## 935 0.18181818 0.81818182
## 937 0.18181818 0.81818182
## 944 0.01886792 0.98113208
## 946 0.01886792 0.98113208
## 947 0.01886792 0.98113208
## 950 0.01886792 0.98113208
## 951 0.01886792 0.98113208

```

```

## 957 0.26666667 0.73333333
## 969 0.53846154 0.46153846
## 970 0.26666667 0.73333333
## 974 0.26666667 0.73333333
## 978 0.18181818 0.81818182
## 981 0.26666667 0.73333333
## 992 0.26666667 0.73333333
## 993 0.26666667 0.73333333
## 994 0.53846154 0.46153846
## 998 0.95121951 0.04878049
## 1001 0.26666667 0.73333333
## 1005 0.18181818 0.81818182
## 1008 0.18181818 0.81818182
## 1013 0.95785441 0.04214559
## 1014 0.95785441 0.04214559
## 1015 0.95785441 0.04214559
## 1016 0.95785441 0.04214559
## 1021 0.95785441 0.04214559
## 1027 0.95785441 0.04214559
## 1032 0.95785441 0.04214559
## 1040 0.26666667 0.73333333
## 1045 0.95785441 0.04214559
## 1048 0.95785441 0.04214559
## 1051 0.95121951 0.04878049
## 1056 0.26666667 0.73333333
## 1058 0.53846154 0.46153846
## 1068 0.95121951 0.04878049
## 1070 0.95121951 0.04878049

```

Valorar resultados.

### Apartado e)

Aplicamos la función cv.tree() a los datos de training y veamos qué hace.

```

model.cv.tree <- cv.tree(model.tree, K = 5)
model.cv.tree

## $size
## [1] 10 9 8 6 5 4 3 2 1
##
## $dev
## [1] 752.4366 721.1024 713.3778 713.3778 713.3778 776.8580 776.8580
## [8] 776.0987 1067.7919
##
## $k
## [1] -Inf 11.00549 13.51258 14.02312 17.78698 37.79043 38.60387
## [8] 46.64903 294.32466
##
## $method
## [1] "deviance"
##
## attr(),"class")
## [1] "prune"           "tree.sequence"

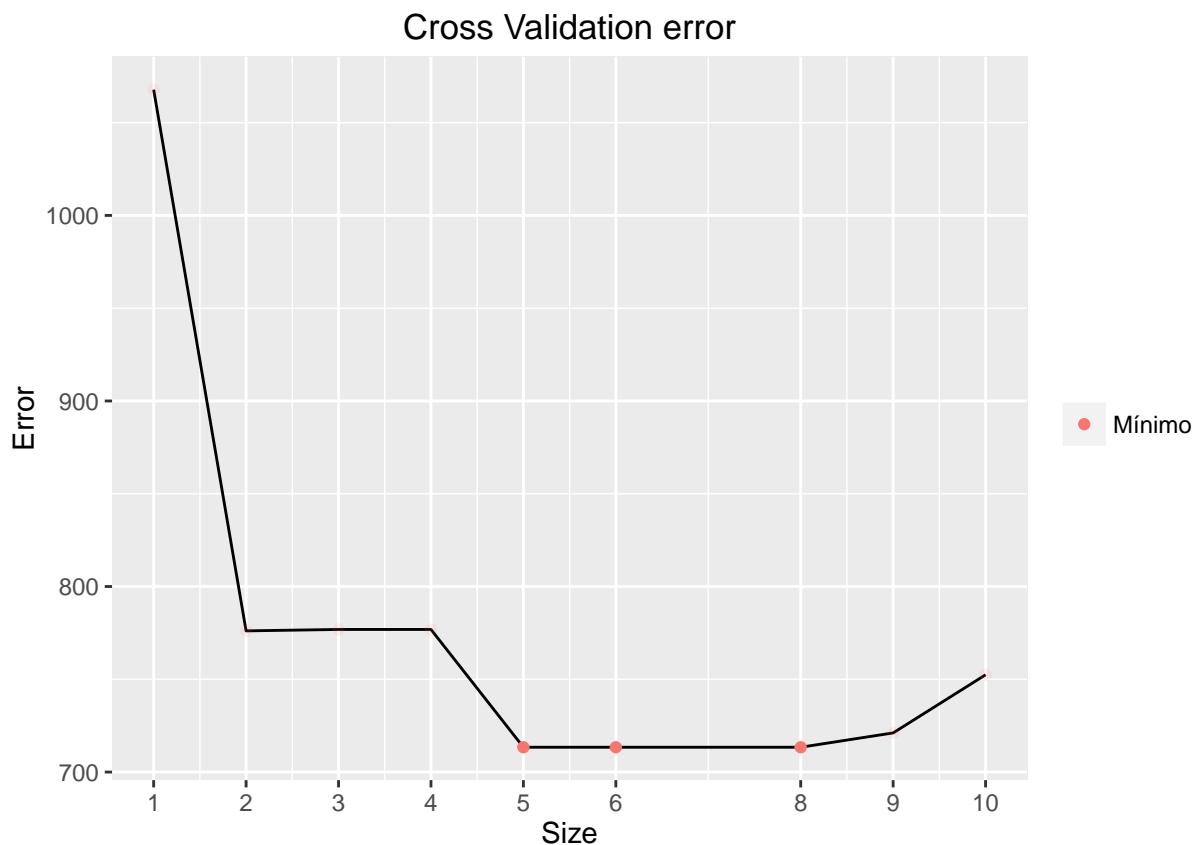
```

## Apartado f) (Bonus-4)

```
error.tree <- data.frame(size = model.cv.tree$size,
                         error = model.cv.tree$dev)

error.tree$alpha <- ifelse(error.tree$error == min(error.tree$error), 1, 0)

ggplot(error.tree, aes(x=size,y=error)) + geom_line() +
  geom_point(aes(colour = "Mínimo", alpha = alpha )) +
  scale_x_continuous(breaks = error.tree$size) +
  guides(alpha=FALSE) +
  theme(legend.title = element_blank()) +
  labs(title = "Cross Validation error",
       x = "Size",
       y = "Error")
```



Podemos ver que el mínimo de error se alcanza en 8, 6, 5.